CS3000
5/20 - tues.

Admin

- HW2 due thurs 9pm
- HW3 out thurs, due 5/30 9pm
- 5/26 no class, no OH
- exam #1 thurs 5/22
- recitation today: practice problems!
- HW1 grades out today

Agenda

1. Binary Trees
2. Binary Search Trees
3. BST operations/run-times

0. Apply

- implement a Queue using 2 stacks
- Enqueue, Dequeue (FIFO)
- Stacks: push, pop, is-empty (LIFO)

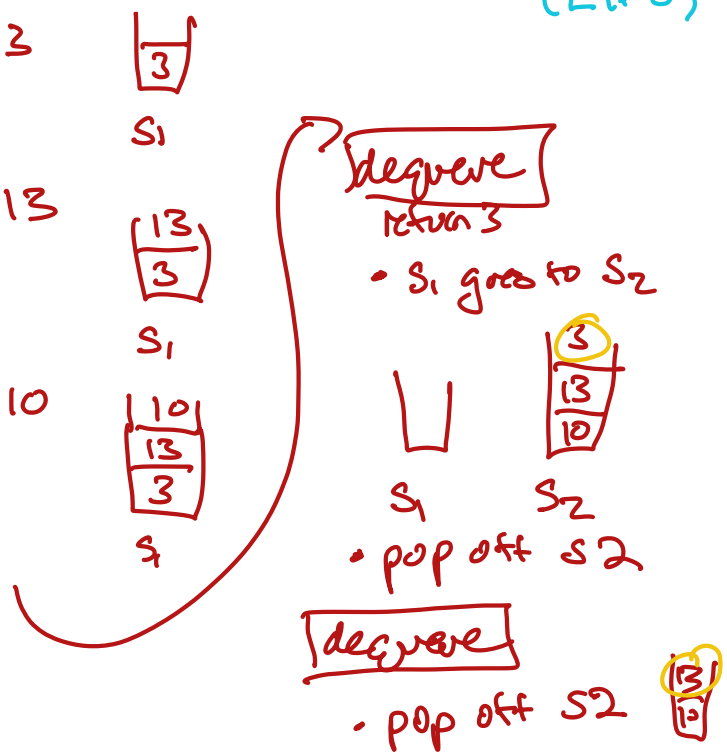
S_1 enqueue

S_2 dequeue

ex enqueue 3

enqueue 13

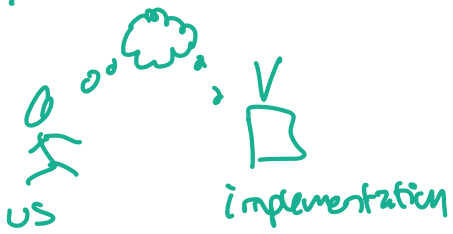
enqueue 10



1. Binary Trees

1. D+C
2. data structure — heaps / queues / stacks — underlying array
3. ...

can we do better?

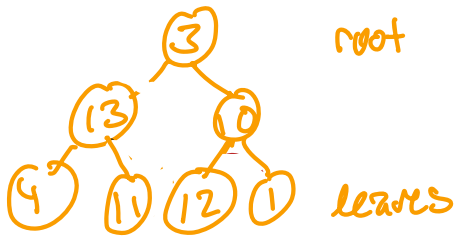


implementation with pointers
- point from one element to another
- attributes on elements

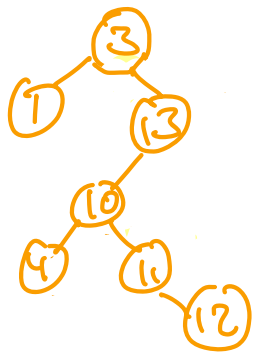
Basic operations:
- insert
- remove
- find

using insert to create...

Binary Tree



Binary SearchTree



common
- ≤ 2 children
- = 1 parent, except root
- connected

diff
- BT fills in each level top to bottom, left to right
- BST puts larger to the right, smaller to the left

if new thing < curr
  go left
otherwise
  go right

* indexing into BT/BST?? no :(:)

* distinct values

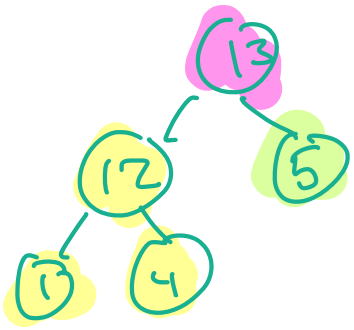
every node has attributes: (n)

n.key	used for comparison
n.left	pointer to left child (or NIL)
n.right	pointer to right child (or NIL)
[n.parent]	pointer to parent (or NIL)

every subtree is also a tree!

First Tree Algorithm: traverse!
- print n.key for every node n
- recursively print everything in left subtree

- print root
- recursively print everything in right subtree



traverse(13)

1, 12, 4, 13, 5

if x == NIL :)

x.right
x.left
and x is NIL :(

"in order traverse"

```
TRAVERSE(x)
  if x ≠ NIL
    TRAVERSE(x.left)
    print x.key
    TRAVERSE(x.right)
```

Runtime $\Theta(n)$
where n is # of nodes

12:35

2. Binary Search Trees

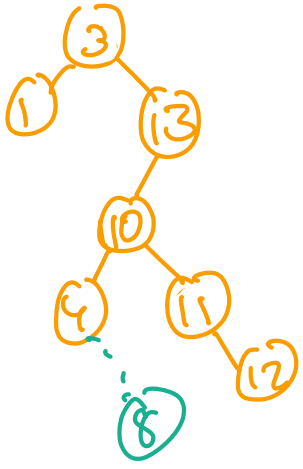
↳ type of binary tree where we maintain order

- n.key for every node n is smaller than right subtree, larger than left sub tree

n.key
n.left
n.right
(n.parent?)

every node has ≤ 2 children

ex



Find on a binary search tree is just binary search, except:
- no array
- no indexing
- start at root

How would I find...
- 13? 13 vs 3 go right 13 vs 13 done!
- 11? 11 vs 3 go right 11 vs 13 go left
  11 vs 10 go right 11 vs 11 done!
- 8? 8 vs 3 go right 8 vs. 13 go left

Key operations
- insert
- remove
- find ★ !!

sim. implementation w/ arrays
- if key is found, return node
- if not found, return NIL

8 vs 10 go left 8 vs. 4 go right return NIL

```
BST-SEARCH(x, k)
1  if x == NIL or k == x.key
2       return x
3  if k < x.key
4       return BST-SEARCH(x.left, k)
5  else
6       return BST-SEARCH(x.right, k)
```

x current root of subtree
k key we're looking for

3. BST Operations and Run-Time

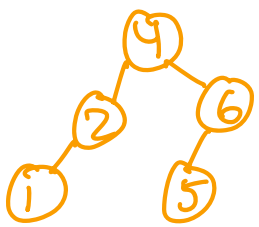
↳ reminder: binary search on a sorted array $\Theta(\lg n)$

binary search on a BST ... ?

insert operation on BST

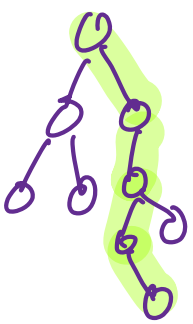
- insert a new node, x
- start at root
- if x.key < root, go left
- otherwise, go right
- new node becomes a leaf

ex insert 4, 6, 2, 5, 1



For insert, search: go left/right depending on x.key vs. root

steps to insert or find



go from root to deepest leaf:
run-time is linear in height of tree
$\Theta(h)$ where h = height
height = # edges from root to deepest leaf

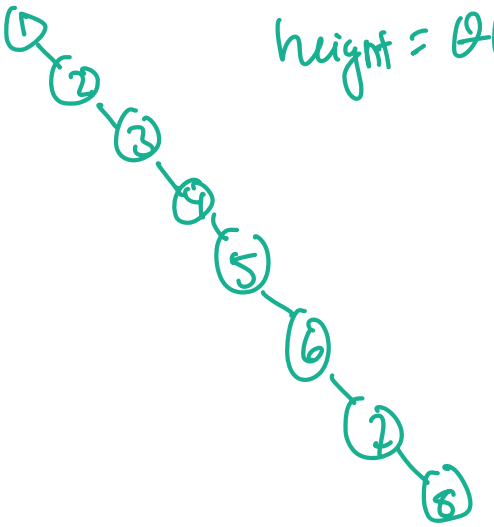
In a heap, $\Theta(h) = \Theta(\lg n)$

In a BST, is height = lg n?

8 leaves in BST

insert values for worst height

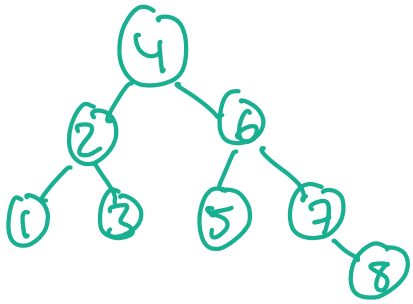
1, 2, 3, 4, 5, 6, 7, 8



height = $\Theta(n)$

insert values for best height

4, 2, 1, 3, 6, 5, 7, 8



height = $\Theta(\lg n)$

Without guarantees of balance:

- wc height $\Theta(n)$
- bc height $\Theta(\lg n)$

Regular BST is used more often!

BSTs can be implemented in a balanced way

- Red-Black tree
- AVL tree

↳ extra step on insert/remove to keep tree in balance

Binary Search

- sorted array?
- BST?

- space (array)
- readability (BST)
- BST maintains order
- better recursive structure (BST)

↳ array: better if data is fixed

BST: better if lots of insert/remove with the search